F2F: reliable storage in open networks

Jinyang Li Frank Dabek
UC Berkeley/MIT MIT
{jinyang, fdabek}@csail.mit.edu

Abstract

A major hurdle to deploying a distributed storage infrastructure in peer-to-peer systems is storing data reliably using nodes that have little incentive to remain in the system. We argue that a node should choose its neighbors (the nodes with which it shares resources) based on existing social relationships instead of randomly. This approach provides incentives for nodes to cooperate and results in a more stable system which, in turn, reduces the cost of maintaining data. The cost of this approach is decreased flexibility and storage utilization. We describe our approach and sketch two applications for which this approach is viable: a cooperative backup system and a Usenet replacement.

1 Introduction

Systems that store data reliably satisfy a fundamental requirement: new copies of data are created faster than existing copies are lost. Satisfying this requirement is difficult in traditional peer-to-peer systems because nodes fail frequently and creating new copies of lost data over limited-capacity wide-area links takes a long time. This combination of frequent node failures and slow data creation can make it expensive, or even impossible, to store large amounts of data durably. Rodrigues and Blake [1] have shown that node failure rate like that observed in Gnutella implies that the bandwidth required to replace failed replicas far exceeds that of a typical cable modem link even if each node only stores 3GB of data.

Since wide-area link capacity is unlikely to dramatically increase in the near future, we have to reduce the node failure rate to increase the storage capacity of peer-to-peer systems. In particular, we are interested in reducing permanent, voluntary node departures: such departures are common in most peer-to-peer systems since these systems offer little or no incentive for nodes to permanently remain in the system and continue to donate resources. Existing incentives used by Bittorrrent and Mojo Nation reward nodes for uploading data but not necessarily for remaining in the system.

The pitiful capacity of the systems analyzed by Blake and Rodrigues is a consequence of users failing to display that good will without proper incentives: after downloading a file, users have no good reason to continue participating in the network (and in fact many good reasons to exit it: resource usage, legal liability, etc.). With no reason to remain in the system, users often leave permanently, forcing any system to make new copies of whatever data the recently departed node is responsible for storing. Even if the node will rejoin the system at a later date (the next time the user wishes to download a file, for example), this behavior results in poor node availability. Poor availability, in turn, leads to reduced capacity in most replication systems since systems can not disambiguate temporary departures from permanent departures and more copies of each data item must be created to guarantee durability.

One solution to the problem of insufficient volunteerism among participants is to eliminate the need for volunteers by operating a closed system: a central authority (usually the system designer) explicitly admits, administers, and guarantees the continued operation of the nodes that make up the system. Many data-intensive peer-to-peer applications have been deployed in such a manner on the PlanetLab testbed. OverCite [23], UsenetDHT [21], and OpenDHT [17] all run only on PlanetLab nodes under the control of their system designers. While solving the problem of high failure rate, this approach limits the system's organic growth and the ultimate scale.

In this paper, we propose a way to structure peer-to-peer storage systems that retains the benefits of an open system (organic growth and scale), but creates an incentive for nodes to increase their reliability. We draw our ideas from social networks, and propose to link nodes in the network based on friendship relationships between the nodes' owners. In real life, people tend to behave more cooperatively towards their friends. Nodes restrict themselves to sharing storage and network resources only with neighbors, but in return can expect that their neighbors (friends) will behave cooperatively. We call this way of structuring networks *f2f* .

This research was conducted as part of the IRIS project (`http://project-iris.net/`), supported by the National Science Foundation under Cooperative Agreement No. ANI-0225660.

Such a network can grow quickly since each node can independently invite other nodes to join the network by enlisting friends (witness the explosive growth of Myspace, Friendster, Orkut, etc.). There are also few permanent node departures in *f2f*: when users, now friends, behave cooperatively towards one another, they are unlikely to permanently leave the system and will maintain higher node availabilities; the alternative is to disappoint a friend by imposing poor service or high overhead on her. As a result of users' good behavior, the system's failure characteristics more closely mirrors that of the underlying hardware than the whims of disinterested users.

f2f offers a new approach to the problem of storing a large amount of data durably on poorly connected, wide-area nodes. The decreased data maintenance overhead and increased capacity provided by *f2f* comes at the cost of decreased flexibility: the system limits resources that can be used to those of a node's neighbors. A *f2f* application can neither use all idle resources nor provide global sharing of common data. We argue that the benefits of having reliable nodes outweighs the attractions of global sharing for many classes of applications. In this paper, we quantify the costs and benefits of such a system construction using two example applications, backup and Usenet.

2 Network and Sharing Model

The network structure we propose, *f2f*, was inspired by the popularity of social networking sites: Friendster, Orkut and Club Nexus. These networks allow users to input the identities of friends and explore the graph created by friendship links by browsing lists of friends' friends.

In *f2f*, each node knows a set of other nodes (neighbors). Links between nodes encode either social relationships or some out-of-band agreements between their corresponding owners that dictate that storage resources will be shared between them. All links are bi-directional. Each application running on a node has a public/private key pair and learns neighbors' public keys using out of band communication.

Most distributed storage systems create a global sharing domain where the resources donated by one node are necessarily available to all nodes. *f2f* captures a more general notion of the users' willingness to cooperate. For example, five mutually trusting friends could agree to run a DHT for backup purposes: the DHT itself spreads all data in the system among all participating nodes. However, if a sixth member joins who is known by some, but not all, of the existing members, the DHT's notion of global sharing no longer matches the users' preferences.

Failure type	Rate (per day per node)	Which trace
disk failure	1/1825	Maxtor datasheet
accidental data loss	1/314	PlanetLab [4]
node departure	1/27	Overnet [16]

Table 1: The permanent failure (f_p) in *f2f* is determined by the rate of disk failure and accident data loss ($1/1825 + 1/314$), while f_p in traditional open networks is dominated by permanent node departures ($1/27$).

3 Benefits

The permanent node failure rate (f_p) places a lower bound on how often replicas must be created: any system must make new copies of data items that permanently leave the system to maintain data durability. We say that a data item is stored durably if it exists in the system and will be able to be retrieved at some time in the future even though the data might not be available now. The most important benefit of *f2f* is a reduction of the permanent failure rate. Because a node stores data only on neighbors that have explicitly agreed to cooperate, we expect f_p in *f2f* to reflect only disk failures or accidental data loss due to operator errors rather than voluntary node departures..

The incidence of disk failure is low; the lifetime of a typical commodity disk is five years [14]. Operator actions that lead to data loss are also rare. On PlanetLab, between October 2004 and September 2005 (356 days), there were 628 data losses on the 555 PlanetLab machines [4]. Most of these losses were caused by the installation of a new version of PlanetLab software. This corresponds to an approximate failure rate of $628/(555 * 356) = 1/314$ failures per day per node for the data stored on each PlanetLab machine. Data loss incidents will also be rare on home users' machines.

In contrast, the rate of data loss caused by permanent node departure in an open network is much higher than either disk or operator failures. We estimate the typical membership changes in an open peer-to-peer network using the 7-day Overnet trace [16] collected in January, 2003. Of the 1198 active nodes in the first 2 days of the trace, 214 of them were never online in the remaining 5 days of the trace. If we consider those 214 nodes as having permanently left the network, the rate is $214/(1198 * 5) = 1/27$ departures per day per node. Table 1 summarizes the permanent failure rates caused by different factors. The f_p in *f2f* is mostly due to disk failures and accidental data loss ($\approx 1/314$), more than an order of magnitude improvement over the node departure rate ($1/27$) in a traditional open network.

A second advantage of *f2f* is a decrease in maintenance bandwidth caused by temporary failures. A system that preserves durability should do no work when nodes temporarily fail. In practice, however, it is impossible to distinguish a transient failure from a permanent node departure by monitoring a node remotely over the network. Most existing systems assume pessimistically that a node has left the system permanently if it is unreachable. In contrast, as neighbors rarely leave the system unannounced, *f2f* nodes assume optimistically that an unreachable neighbor has suffered a temporary failure. A node can rely on out-of-band information from the neighbor's owner to learn about rare permanent node departures.

Without the ability to distinguish transient failures from permanent failures, a system ends up maintaining data availability in order to achieve durability, keeping around a total of approximately $\frac{2}{a} \times r$ replicas [4], where a is the host availability and r is the initial replication level. Overnet nodes' average availability is 0.353. Therefore, a system needs to incur an extra 5.6 times the storage and bandwidth overhead than *f2f* as a result of creating redundant replicas when faced Overnet node availability.

In summary, *f2f* incurs $s \cdot f_p \cdot r$ communication overhead per node to ensure data is stored durably in the system, where s is the amount of unique data stored on each node and $f_p \approx 1/314$ (assuming desktop machines have the same data loss rate as that in PlanetLab). For $r = 2$, $s = 1$ TB, the overhead is only 6 GB of data transfer per node per day (or equivalently, 0.07 MB/node/second) which can be handled comfortably even with a cable modem link. On the contrary, in systems designed to work in traditional open networks such as Overnet, each node requires $s \cdot f_p \cdot 2r/a$ bandwidth where $f_p = 1/27$ and $a = 0.353$. For $r = 2$, $s = 1$ TB, the overhead is 419 GB of data transfer per node per day (or equivalently, 4 MB/node/second), roughly 70 times the maintenance bandwidth needed in *f2f*. This bandwidth is an average over a long period of time, across multiple failures at a node. We can apply techniques in [22] to reduce the peak bandwidth consumption.

4 Case Study I: backup

To illustrate how applications designed for *f2f* depart from previous proposals for a traditional open peer-to-peer network, we sketch the design of a distributed backup application, BlockParty, and compare it to existing backup systems (Pastiche [2] and Venti-DHash [20]).

BlockParty provides an affordable, off-site backup service for home users. It is an alternative to purchasing a redundant disk (expensive and complicated) or paying for a network backup service (expensive). We expect users to backup unique, difficult to replace data such as digital photo libraries or music collections. Recovering a large amount of data from this system will be tedious (months are required to recover $1T$ data over a cable modem link), but the user can selectively download important data first while archives are slowly restored.

BlockParty breaks the data to be backed up into chunks and distributes each chunk to one or more neighbor machines depending on the desired replication level r; chunks are preferentially sent to the neighbor with the most free space. Data is only sent to neighbors, since these nodes are owned by individuals we believe will act cooperatively. In BlockParty, the existence of a link in the underlying *f2f* network implies a storage contract: neighbors agree to share use each other's spare capacity for backup. The existence of social connections between neighbors gives a node reason to believe that these contracts will be honored: a neighbor's incentive not to leave the system or destroy backed up copies of data is avoiding his friend's ire. We assume that neighbors might be negligent or sloppy, but that they are cooperative. To ensure storage balance, the BlockParty software at a node dedicates at least as much space to storing other nodes' backups as the node wishes to use on other nodes.

Venti-DHash is a backup system based on a distributed hash table [5]. The Venti-DHash software running on each machine converts data to be backed up into small blocks that are organized into a hash tree structure stored in the DHT. Pastiche uses a modified version of Pastry [18] to find, for each participant, a "buddy" that stores similar files; the participant then sends the contents of its disk to the one buddy (instead of spreading the contents of the disk across many machines as Venti-DHash does). The buddy is chosen from all participating machines, similar to the proposal in [12].

4.1 Maintenance Traffic

A BlockParty node periodically queries neighbors that store data to determine if the data is intact; when a neighbor does not respond, BlockParty can delay making a new copy of the data since the friendship relationship allows the node to safely assume that the failure is temporary. After a long timeout, or perhaps out-of-band verification of a hardware failure, the system sends a new copy of the data to an alternative neighbor. In contrast, Venti-DHash, must create more copies of data due to a higher permanent node departure rate and in response to node unavailability.

As the analysis in Section 3 shows, each Venti-DHash node needs to consume 4 MB/second in order to maintain 2 copies of his 1 TB of data durably in an open network that has failure properties similar to the Overnet trace. BlockParty, by contrast, requires

$0.07MB/second/node$.

4.2 Ensuring Cooperative Node Behavior

While the primary benefit of a *f2f* system is reduced maintenance traffic, the use of social links also provides nodes an incentive for good behavior other than not leaving the system. BlockParty users can expect that their neighbors, chosen to reflect social links, will not deliberately delete data or modify the BlockParty software to offer less storage to their neighbors.

Venti-DHash, by contrast, assumes that *all* users running DHT nodes are cooperative. There is no mechanism in Venti-DHash to ensure that a node donates the same amount of storage as it consumes in the system.

Pastiche assumes that all nodes are malicious or selfish unless proven not be so. It uses the Samsara system [3] to enforce a storage invariant: the amount of data that a node stores into the system equals the amount of data the system stores on a node. If a node x wishes to store data on y, it has to store y's data or claim (randomly generated data) in return. Samsara punishes uncooperative nodes that delete or lose others' data by deleting those nodes' data. Each node periodically pings its buddies to obtain a cryptographic proof that they actually store the backed up data. If a buddy is unavailable or unable to provide the proof, the querier probabilistically deletes his data based on the assumption that the buddy failing the query has deleted the data in a selfish attempt to gain storage without donating an equal amount of storage or becoming unavailable purposefully to avoid work.

Samsara's punishment scheme might not work well in practice since it does not provide all nodes an incentive not to leave or delete data. In Samsara, if a node y receives data from x, the only way x can penalize y for deleting his data is to delete y's claim; claims are random data which y is unlikely to miss. Only if claims are forwarded to form a cycle is a node penalized by losing his backed up data. This scheme also has a particularly adverse effect for cooperative nodes suffering a disk failure; a failed Samsara node begins a race against the clock to fetch its lost data before the buddies holding that data identify the node attempting to restore its data as a cheater.

We've used Samsara as an example, but any system that assumes that nodes are malicious is likely to be as complicated and costly as Samsara; BlockParty's benefits arise from changing the rules of the game by operating within a node's cooperative domain. The cost of playing under more generous rules of node cooperation is a limitation on what resources a node may access. We now evaluate how this limitation impacts BlockParty's ability to efficiently use storage resources.

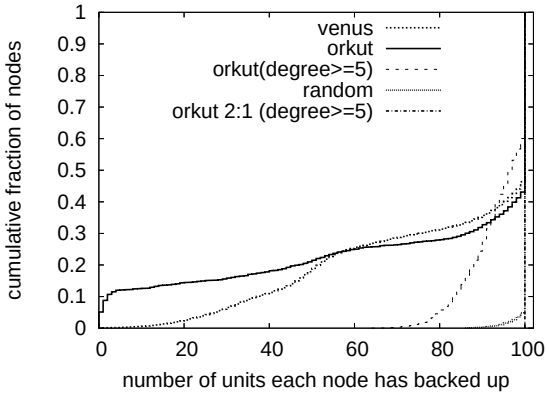

Figure 1: The distribution of amount of successfully backed up data among nodes in *venus*, *orkut* and a random graph. We ensure each node has degree at least 5 in the *orkut(degree>= 5)* graph by requiring nodes with less than 5 neighbors to befriend some random nodes. *orkut 2:1 (degree>= 5)* shows the results when each node donates 200 units storage space to back up 100 units data. All nodes succeed in backing up all their data in this case.

4.3 Storage Efficiency

BlockParty needs to allocate data to neighbors to maximize the total amount of data backed up. The allocation must respect nodes' storage constraints. Unfortunately, even with global information and coordination, computing the best allocation is NP-complete [1]. Furthermore, in certain worst case graphs, even the best possible space utilization is low. Consider a simple graph where node a has 10 neighbors, each with no neighbors other than a. Assume each node needs to back up 10 unit data and contributes 10 unit free storage space. Node a can back up all of its data (it sends 1 unit to each friend), but a's 10 neighbors must share a's storage: each neighbor backs up 1 unit on a. In this graph, only 20 units of total data are backed up, resulting in $\frac{20}{110} = 18\%$ space utilization.

Fortunately, with real life social graphs, the simple strategy of choosing a neighbor with the most free space to back up a chunk of data works well. We simulate BlockParty's performance using two data sets: *orkut* which we obtained by crawling Orkut and *venus* which is a complete set of friendship links provided by a large online dating site. The *orkut* and *venus* social graphs are illustrative, in practice, the set of people one is willing to place in an Orkut friend list, might not be the same set that one is willing to trust with important data. The *orkut* data set consists of $2,363$ users with a median node degree of 7. The *venus* data set includes $39,783$ users with a median

[1]We can reduce maximum independent set to the backup problem where each node tries to swap 1 unit of data with one of its neighbors.

node degree of 2. In our simulations, each node attempts to backup 100 units of and donates 100 units of free space. At each step, a random node copies 1 unit of its remaining data on the neighbor with the most available space while there is still free space on its neighbors.

The total space utilization of BlockParty on the *orkut* and *venus* graphs is 78% and 81% respectively. Nodes succeed in backing up different amounts of data depending on their positions in the network. Figure 1 shows the cumulative distribution of the amount of data each node is able to back up on its neighbors. More than half of the nodes manage to back up all data in both graphs. However, roughly 20% of the nodes only manage to back up less than half of the target amount of data.

In Figure 1, we also compare *orkut* result to that obtained in a random graph with the same number of nodes and links as *orkut*. The space utilization on a random graph is much higher (98%) and all nodes manage to back up at least 80 units. The inefficient space utilization in *orkut* is a result of some nodes having few neighbors; this is partly because we are unable to crawl the *orkut* population completely, leaving many nodes' neighborhood unexplored. In particular, 25% *orkut* nodes have 2 or less neighbors while few nodes ($< 4\%$) in the random graph have less than 5 neighbors. One simple yet effective optimization is to enforce a minimal node degree by encouraging people to make more friends. As Figure 1 shows, if we enforce all *orkut* nodes to have at least 5 friends, the overall space utilization increases to 93%. Furthermore, 95% of the nodes manage to back up at least 80 units of their data. To further increase the chance of achieving full backups, we can sacrifice storage utilization by demanding each node donate more resource than it consumes. Figure 1 shows that if each node donates 2 times more space than it consumes, in addition to having at least 5 neighbors, all nodes successfully back up all their data.

5 Case Study II: Usenet

Usenet [9] is a large distributed bulletin board service. It is currently connected as a mesh of servers. Usenet uses a flood-fill algorithm to propagate and therefore replicate all articles to all servers that carry a "full feed". This is very expensive in terms of bandwidth and disk storage, requiring more than 100Mbps bandwidth to replicate 2TB of data daily [21]. UsenetDHT [21] is a re-design of Usenet that organizes servers into a DHT. UsenetDHT replaces local article storage with shared storage provided by the DHT. All articles are replicated only twice by the DHT (instead of n times where n is the number of nodes in the system) and each server only downloads the article that users actually read from the DHT (instead of every article posted).

Since the links between existing Usenet servers already reflect an out-of-band peering agreement, we propose to replicate articles along these links: each article is stored on the server where it was originally posted and one of that server's immediate neighbors. Meta-data and control-data are flooded to all servers; each server maintains a list of all articles that includes, for each article, the IP addresses of the source and replica nodes.

This scheme retains same benefits of UsenetDHT (operating in a closed system like PlanetLab): increased capacity and reduced network requirements when compared to Usenet. The system-wide storage overhead of this scheme is twice the amount of daily storage required to store unique articles ($2 \times$ 2TB) instead of $2n$, where n is the number of full feed servers. The system's bandwidth consumption is proportional to the data read at each node (typically 1% of all articles) instead of the total amount of data posted. NewsCache [7] reduces resource consumption of leaf nodes by caching news articles that are read instead of obtaining the full feed, while UsenetDHT and our proposal both aim to reduce the bandwidth consumption of backbone and non-leaf nodes. Usenet in *f2f* also spends less bandwidth in ensuring data durability as it does not need to create an extra $2r/a$ of replicas in response to transient failures as UsenetDHT does.

6 Discussion

f2f is not applicable to all types of peer-to-peer applications. It is especially suitable for data intensive applications that do not require global visibility or global sharing. For example, cooperative backups, as well as private file sharing, do not depend on global participation: a small number of stable neighbors suffice to back up a node's data. The presence of hundreds or even millions more participating nodes are not necessarily helpful. Usenet on *f2f* demonstrates a different type of application suitable for *f2f*, one which has localized storage and relies on a separate mechanism (i.e. flood-fill of meta data) to achieve data visibility.

The underlying social graph in *f2f* is application specific since a node's owner may cooperate with different groups in the context of different applications. For example, a user may choose to back up her data on close friends but engage in private sharing with a larger circle of people.

7 Related Work

Social networks, also known as "small world" networks have been well studied in the past [10] for their low network diameter and the ability to route queries quickly [11]. They are used to enhance existing online

reputation systems (e.g. [8][19]) by identifying reputation based on position in the social networks or filtering friends' ratings to avoid collusion. Reputation systems are also used to predict the likelihood of a random node behaving correctly given its past performance observed by others. Many systems use social networks to predict or ensure nodes' trustworthiness, e.g. SPROUT [13], Maze [24], Turtle [15]. The idea of having a node explicitly pick out other trustworthy nodes also resembles that in SPKI/SDSI [6] and PGP certification chain. In comparison, *f2f* uses social networks to provide incentives for nodes to contribute to the system and thus obtain reliable storage.

8 Conclusion

This paper has outlined a new way to structure peer-to-peer systems. We argue that a user should explicitly identify, based on existing social relationships, the nodes with which it wishes to share storage and network resources. Systems structured in this way require less bandwidth to maintain data durability compared to open systems because nodes have an incentive to remain in the system. This approach is not suitable to all applications, but a number of applications can be built on this style of overlay: we have designed a a cooperative backup system and Usenet replacement.

References

[1] C. Blake and R. Rodrigues. High availability, scalable storage, dynamic peer networks: Pick two. In *9th Workshop on Hot Topics in Operating Systems*, May 2003.

[2] L. P. Cox, C. Murrary, and B. Noble. Pastiche: Making backup cheap and easy. In *5th Symposium on Operating Systems Design and Implementation (OSDI 2002)*, 2002.

[3] L. P. Cox and B. Noble. Samsara: Honor among theives in peer-to-peer storage. In *19th ACM Symposium on Operating Systems Principles(SOSP 2003)*, 2003.

[4] F. Dabek. *A Distributed Hash Table*. PhD thesis, Massachusetts Institute of Technology, Oct. 2005.

[5] F. Dabek, M. F. Kaashoek, J. Li, R. Morris, J. Robertson, and E. Sit. Designing a DHT for low latency and high throughput. In *1st NSDI*, March 2004.

[6] C. Ellison, B. Frantz, B. Lampson, R. Rivest, B. Thomas, and T. Ylonen. Spki certifi cate theory. RFC 2693, Network Working Group, 1986.

[7] T. Gschwind and M. Hauswirth. NewsCache: A high-performance cache implementation for Usenet news. In *1999 USENIX Annual Technical Conference*, pages 213–224, June 1999.

[8] T. Hogg and L. Adamic. Enhancing reputation mechanisms via online social networks. In *Proceedings of the 5th ACM conference on Electronic Commerce*, 2004.

[9] B. Kantor and P. Lapsley. Network news transfer protocol. RFC 977, Network Working Group, Feb. 1986.

[10] J. Kleinberg. The small-world phenomenon: An algorithmic perspective. In *32nd Symposium on Theory of Computing*, 2000.

[11] D. Liben-Nowell. *An Algorithmic Approach to Social Networks*. PhD thesis, Massachusetts Institute of Technology, June 2005.

[12] M. Lillibridge, S. Elnikety, A. Birrel, and M. Burrows. A cooperative internet backup scheme. In *USENIX Annual Technical Conference*, 2003.

[13] S. Marti, P. Ganesan, and H. Garcia-Molina. DHT routing using social links. In *3rd International Workshop on Peer-to-Peer Systems (IPTPS 2004)*, 2004.

[14] Maxtor diamondmax 16 datasheets. `http://www.maxtor.com/`.

[15] B. C. Popescu, B. Crispo, and A. S. Tanenbaum. Safe and private data sharing with turtle: Friends team-up and beat the system. In *Proc. 12th Cambridge International Workshop on Security Protocols*, 2004.

[16] S. S. Ranjita Bhagwan and G. Voelker. Understanding availability. In *Proceedings of the 2003 International Workshop on Peer-to-Peer Systems*, Feb. 2003.

[17] S. Rhea, B. Godfrey, B. Karp, J. Kubiatowicz, S. Ratnasamy, S. Shenker, I. Stoica, and H. Yu. OpenDHT: A public DHT service and its uses. In *Proceedings of ACM SIGCOMM*, Aug. 2005.

[18] A. Rowstron and P. Druschel. Pastry: Scalable, distributed object location and routing for large-scale peer-to-peer systems. In *Proceedings of the 18th IFIP/ACM International Conference on Distributed Systems Platforms (Middleware 2001)*, Nov. 2001.

[19] J. Sabater and C. Sierra. Social regret, a reputation model based on social relations. In *ACM SIGecom Exchanges*, 2002.

[20] E. Sit, J. Cates, and R. Cox. A DHT-based backup system. In *IRIS Student Workshop*, 2003. `http://project-iris.net/isw-2003/papers/sit.pdf`.

[21] E. Sit, F. Dabek, and J. Robertson. UsenetDHT: A low overhead Usenet server. In *3rd International Workshop on Peer-to-Peer Systems*, Feb. 2004.

[22] E. Sit, A. Haeberlen, F. Dabek, B.-G. Chun, H. Weatherspoon, R. Morris, M. F. Kaashoek, and J. Kubiatowicz. Proactive replication for data durability. In *5rd International Workshop on Peer-to-Peer Systems (IPTPS 2006)*, 2006.

[23] J. Stribling, I. G. Councill, J. Li, M. F. Kaashoek, D. R. Karger, R. Morris, and S. Shenker. OverCite: A cooperative digital research library. In *4th International Workshop on Peer-to-Peer Systems (IPTPS 2005)*, Feb. 2005.

[24] M. Yang, H. Chen, B. Y. Zhao, Y. Dai, and Z. Zhang. Deployment of a large-scale peer-to-peer social network. In *USENIX WORLDS*, 2004.